042390.P2445 OIPE JC41 MAR 06 2002 PATENT & TRADEMARK OFFICE COPY COPY OF PAPERS ORIGINALLY FILED Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:
Alexander Peleg, et al.

Serial No. 08/522,067

Filed: August 31, 1995

For: A Method for Performing Multiply-Add Operations on Packed Data

Examiner: E. Moise

Art Unit: 2306

Assistant Commissioner of Patents and Trademarks
Washington, D.C. 20231

APPEAL BRIEF UNDER 37 C.F.R. §1.192

Sir:

The Applicant hereby submits this Brief in support of its appeal from a final decision of the Examiner in the above-captioned case.

(1) REAL PARTY IN INTEREST

Intel Corporation of Santa Clara, CA.

(2) RELATED APPEALS AND INTERFERENCES

None.

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage in an envelope addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231

on October 17, 1997
Date of Deposit

David Roberts
Name of Person Mailing Correspondence

David Roberts 10/7/97
Signature Date

(3) STATUS OF CLAIMS

In this application, Claim(s) 24-26 and 28-33 are pending, while Claim(s) 1-23 and 27 were canceled without prejudice.

Claim(s) 24-26 and 28-33 stand rejected under 35 U.S.C. § 103 as being unpatentable over Ando et al. (US Patent No. 4,771,379) in view of J. Shipnes "Graphics Processing with the 88110 RISC Microprocessor", IEEE, 1992, pp. 169-174 ("Shipnes").

The Applicant appeals the Examiner's decision on Claim(s) 24-26 and 28-33 with respect to the rejection under Sections 103.

(4) STATUS OF AMENDMENTS

A response after final was filed on July 17, 1997 under 37 C.F.R. § 1.116 that requested amendments to the specification and proposed amendments to the claims. With regard to the proposed amendments to the claims, Applicant requested the Examiner enter the proposed claim amendments (discussed during an interview) by Examiner's amendment only if the proposed amendments placed the application in condition for allowance. In an advisory action mailed August 4, 1997, the Examiner indicated the response after final was considered and the amendments to the claims were not be entered.

The claims found in the Appendix of this Appeal Brief reflect the claims as they are understood by the Applicant to stand at the date of this appeal.

(5) SUMMARY OF INVENTION

The invention generally relates to the field of computer systems, and more specifically, to the area of packed data instructions[1] executable by a processor in a computer system (see Figure 1, reference characters 101 and 109).

[1] In typical computer systems, processors are implemented to operate on values represented by a large number of bits (e.g., 64) using instructions that produce one result. For example, the execution of an add instruction will add together a first 64-bit value and a second 64-bit value and store the result as a third 64-bit value. However, multimedia applications (e.g., 2D/3D graphics, image processing, video

With reference to the claims at issue, the invention is a single packed data instruction whose execution causes at least two independent multiply-add operations on packed data inputs. By way of example, the operation performed by a processor in response to executing one embodiment of multiply-add instruction is shown below in Tables 3a and 3b duplicated from Applicant's specification -- Table 3a shows a simplified representation of the multiply-add instruction, while Table 3b shows a bit level example of the multiply-add instruction. (pg. 12, line 1 - pg. 13, line 10; and pg. 27, line 5 - pg. 29, line 7):

Multiply-Add Source1, Source2

A_1	A_2	A_3	A_4	Source1
B_1	B_2	B_3	B_4	Source2
		=		
$A_1B_1+A_2B_2$		$A_3B_3+A_4B_4$		Result1

Table 3a

---

compression/decompression, recognition algorithms and audio manipulation) require the manipulation of large amounts of data which may be represented in a small number of bits. For example, graphical data typically requires 8 or 16 bits and sound data typically requires 8 or 16 bits. Each of these multimedia applications requires a number of operations (e.g., an add, compare and shift operation).

To improve efficiency of multimedia applications (as well as other applications that have the same characteristics), certain processors provide packed data formats (e.g., see Figure 4, reference characters 401-403). A packed data format is one in which the bits typically used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value. For example, a 64-bit register may be broken into two 32-bit elements, each of which represents a separate 32-bit value. In addition, these prior art processors provide instructions for separately manipulating each element in these packed data types in parallel. For example, a packed add instruction independently adds together corresponding data elements from a first packed data and a second packed data. Thus, if a multimedia algorithm requires a loop containing five operations that must be performed on a large number of data elements, it is desirable to pack the data and perform these operations in parallel using packed data instructions. In this manner, these processors can more efficiently process multimedia applications.

11111111 11111111	11111111 00000000	01110001 11000111	01110001 11000111
Multiply	Multiply	Multiply	Multiply
00000000 00000000	00000000 00000001	10000000 00000000	00000100 00000000
↓	↓	↓	↓
32-Bit Intermediate Result 4	32-Bit Intermediate Result 3	32-Bit Intermediate Result 2	32-Bit Intermediate Result 1

Add	Add
11111111 11111111 11111111 00000000	11001000 11100011 10011100 00000000

Table 3b

where Source1, Source2 and Result1 represent registers; where A_{1-4} and B_{1-4} represent independent data elements respectively stored in the Source1 and Source2; and where $A_1B_1+A_2B_2$ and $A_3B_3+A_4B_4$ represent data elements stored in Result1.

Thus, the described embodiment of the multiple-add instruction multiplies together corresponding 16-bit data elements stored in Source1 and Source2 generating four 32-bit intermediate results. (pg. 13, lines 2-4) These 32-bit intermediate results are summed by pairs producing two 32-bit results that are packed into their respective elements of a packed result. Id. As further described in the application, alternative embodiment may vary the number of bits in the data elements, intermediate results, and results. In addition, alternative embodiment may vary the number of data elements used, the number of intermediate results generated, and the number of data elements in the resulting packed data.

Figure 8 illustrates an exemplary circuit for executing, among other things, the instruction described above in Tables 3a and 3b (see also pages 29, line 9 - page 31, line 8). As described in pages 31-41, the multiply-add instruction is useful for a number of different applications, including the multiplication of complex numbers, multiply-accumulate operations, dot product operations, and discrete cosine transforms.

(6) <u>ISSUES</u>

In the Final Office Action mailed May 29, 1997, the Examiner has:

(1) continued a rejection under Section 103 of Claim(s) 24-26 and 28-33 based on <u>Ando and Shipnes</u>.

The question presented on this Appeal is:

(1) Whether the combination of <u>Ando and Shipnes</u> renders the claimed invention obvious?

(7) <u>GROUPING OF CLAIMS</u>

Claims 24-26 and 28-33 do not stand or fall together. Rather, the claims are to be grouped as follows:

(i) 24, 26, 29, 31 and 32 with respect to the 35 U.S.C. 103 rejection; and

(ii) 25, 28, 30, and 33 with respect to the 35 U.S.C. 103 rejection.

Each claim group above is considered separately patentable for reasons provided below.

(8) <u>ARGUMENT</u>

Question 1 -- Whether the combination of <u>Ando and Shipnes</u> renders the claimed invention obvious?

a) <u>The Combination of Ando and Shipnes</u>

The multiply-accumulate instruction of the cited combination **<u>always</u>** sums the results **of all** of the multiplications <u>and</u> adds **a previously stored accumulation value** to generate a **<u>single</u>** result value (In Ando, see ALU 32 and ACC 33 of Figure 1; ALU 323 and ACC (accumulator) 33 of Figure 2; ALU 322 and ACC 33 of Figure 3). An example of the operation of the multiply-accumulate instruction of Ando is shown in Table 2 of

the specification (duplicated below with emphasis added), where the instruction is performed on the data values A_1, A_2, B_1 and B_2 accessed from Source1-4, respectively (pages 4, line 19 -page 5, line 5).

Source1		Source3
A_1	Multiply	A_2
Source2		**Source4**
B_1		B_2

= Result1

$A_1 \cdot B_1 + A_2 \cdot B_2$ + **Accumulation Value**

The Examiner acknowledges that the "prior art of record does not teach or suggest a method where the multiply-accumulate instruction does not carry forward the accumulation value." (Final Office Action amiled May 29, 1997, paragraphs 7 and 8).

b) The Combination of Ando & Shipnes does not teach the Claimed Invention

In contrast, the claimed instruction **does not** sum the results of all of the multiplications and **does not** carry forward an accumulation value. While the invention is only limited by the claims, Table 3a from Applicant's Specification is again duplicated for exemplary purposes.

Multiply-Add Source1, Source2

A_1	A_2	A_3	A_4	Source1
B_1	B_2	B_3	B_4	Source2
=		=		
$A_1B_1+A_2B_2$		$A_3B_3+A_4B_4$		Result1

Table 3a

Each of Applicant's independent claims requires either: 1) that execution of the instruction is completed **without summing/accumulating** the results of the multiply-add

operations and without **adding an accumulation value** (claim 24 - "without adding said first and second data elements"; and claim 26 - "without summing said plurality of result data elements"); or 2) that a packed result containing the two **unaccumulated** data elements is stored as an operand for use by another instruction (claims 25 and 28).

c) <u>The Combination of Ando Shipnes does not make the Claimed Instruction obvious</u>

The cited combination does not make the claimed instruction obvious because zeroing the accumulation value of the cited combination is costly considering the purpose for which the multiply-accumulate instruction was designed. Since video and audio requires the manipulation of a large number of small values (e.g., separate data elements for each pixel of each image in motion video), multimedia applications tend to be repetitive in nature because the same operations need to be performed on all of the small values. The multiply-accumulate instruction of Ando was designed for efficiently performing the large number of repetitive multiplication and accumulation operations required by many multimedia algorithms. No where in Ando or Shipnes is there any indication that the instruction is useful for other purposes, much less an efficient technique for zeroing the accumulation value.

In contrast, since the claimed instruction does not carry forward a single accumulation value or accumulate the results, the claimed instruction can be used for performing other operations required by certain multimedia applications. For example, the multiplication of two complex numbers can be performed in response to execution a single one of the claimed instructions (See Table 6a duplicated from Applicant's specification)

Multiply-Add Source1, Source2

r_1	i_1	r_1	i_1	Source1
r_2	$-i_2$	i_2	r_2	Source2
=		=		
Real Component: $r_1r_2-i_1i_2$		Imaginary Component: $r_1i_2+r_2i_1$		Result 1

Table 6a

The cited combination cannot perform the function of multiplying together two complex numbers using one multiply-accumulate instruction. To illustrate, multiplying two complex numbers together using the multiply-accumulate instruction of the suggest combination would require: 1) zeroing out the accumulation value; 2) performing the multiplication of the real component (see Table 10); 3) zeroing out the accumulation value; and 4) performing the multiplication of the imaginary component. In each of these operations, the ALU, which is devoted to the accumulation value, is superfluous hardware and extra instructions are needed to zero the accumulation value. These extra instructions would otherwise have been unnecessary.

r_1		i_1
	Multiply	
r_2		$-i_2$
	=	Result
$r_1 \cdot r_2 - i_1 \cdot i_2 + 0$		

Table 10

By way of a particular example, performing the large number (e.g., a hundred) of complex multiplication operations required by a typical multimedia algorithm using the prior art multiply-accumulate instruction would require generating the real and imaginary

components using different multiply-accumulate instructions and zeroing the accumulation value a large number of times; whereas performing the same complex multiplications using the claimed instruction does not require this zeroing and both the real and complex components are generated in a single instruction.

In addition to the multiplication of complex numbers, the versatility of the claimed instruction allows it to also be use for multiply accumulate operations (see Applicant's specification pg. 34 - 37), dot product operations (see Applicant's specification pg. 37 - 39), discrete cosign transform operations (see Applicant's specification pg. 39-41), etc.

CONCLUSION

For all of the foregoing reasons, this Board is respectfully requested to remand this application to the Examiner for reconsideration consistent with an order that the Examiner pass this case to issuance unless a proper rejection to the claims can be made.

FEE FOR FILING A BRIEF IN SUPPORT OF APPEAL

Enclosed is a check in the amount of $310.00 to cover the fee for filing of a brief in support of an appeal required under 37 C.F.R. 1.17(f) and 1.192.

CHARGE OUR DEPOSIT ACCOUNT

Please charge any shortage to our Deposit Account No. 02-2666.

Respectfully submitted,

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
ORIGINAL SIGNED BY

DMD 10/17/97

Date: ____________, 1997

Daniel M. De Vos
Reg. No. 37,813

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, California 90025-1026
(408) 720-8598

(9) APPENDIX

24. A computer-implemented method responsive to the execution of a single instruction comprising the steps of:

A) multiplying together a first value and a second value to generate a first intermediate result;

B) multiplying together a third value and a fourth value to generate a second intermediate result;

C) multiplying together a fifth value and a sixth value to generate a third intermediate result;

D) multiplying together a seventh value and an eighth value to generate a fourth intermediate result;

E) adding together said first intermediate result and said second intermediate result to generate a first data element in a first packed data;

F) adding together said third intermediate result and said fourth intermediate result to generate a second data element in said first packed data;

G) storing said first packed data, and

H) completing execution of said single instruction without adding said first and second data elements.

25. In a computer system, a method for manipulating a first packed data and a second packed data responsive to the execution of a single instruction, said first packed data including A_1, A_2, A_3, and A_4 as data elements, said second packed data including B_1, B_2, B_3, and B_4 as data elements, said method comprising the steps of:

performing the operation $(A_1 \times B_1) + (A_2 \times B_2)$ to generate a first data element in a third packed data;

performing the operation $(A_3 \times B_3) + (A_4 \times B_4)$ to generate a second data element in said third packed data;

storing said third packed data for use as an operand to another instruction.

26. In a computer system having stored therein a first packed data and a second packed data each containing initial data elements, each of said initial data elements in said first packed data having a corresponding initial data element in said second packed data, a method for performing multiply add operations in response to a single instruction, said method comprising the steps of:

multiplying together said corresponding initial data elements in said first packed data and said second packed data to generate corresponding intermediate data elements, said intermediate data elements being divided into a number of sets;

generating a plurality of result data elements, a first of said plurality of result data elements representing the sum of said intermediate result data elements in a first of said number of sets, a second of said plurality of result data elements representing the sum of said intermediate result data elements in a second of said number of sets; and

completing execution of said single instruction without summing said plurality of result data elements.

28. In a computer system, a method for manipulating a first packed data and a second packed data responsive to the execution of a single instruction, said first packed data including A_1, A_2, A_3, and A_4 as data elements, said second packed data including B_1, B_2, B_3, and B_4 as data elements, said method comprising the steps of:

multiplying together A1 and B1 to generate a first intermediate result;

multiplying together A2 and B2 to generate a second intermediate result;

multiplying together A3 and B3 to generate a third intermediate result; and

multiplying together A4 and B4 to generate a fourth intermediate result;

performing in parallel the following steps:

adding together said first intermediate result and said second intermediate result to generate a first data element in a third packed data; and

adding together said third intermediate result and said fourth intermediate result to generate a second data element in said third packed data; and

saving said third packed data for use as an operand to another instruction.

29. The method of claim 24 further including the steps of:

accessing said first, third, fifth, and seventh values from a storage area; and

writing said first packed data over said first, third, fifth, and seventh values in said storage area.

30. The method of claim 25, further includes the steps of:

accessing said first packed data from a register; and

writing said third packed data over said first packed data in said register.

31. The method of claim 26, further includes the steps of:

storing said plurality of result data elements as a third packed data for use as an operand to another instruction.

32. The method of claim 26, further includes the steps of:

accessing said first and second packed data from a register; and

writing said plurality of result data elements over said first packed data in said register.

33. The method of claim 28, further includes the steps of:

accessing said first and second packed data from a storage area; and

writing third packed data in said storage area.